ECE 375 LAB 5

Large Number Arithmetic

Lab Time: Friday 4-6

Faaiq Waqar

Jordan Brown

Introduction

Lab 5 had us construct subroutines that performed arithmetic operations on numbers larger than 8 bits. Managing the carry bit is very important to these operations and they are done 4 bits at a time. These operations are separated into subroutines and individually verified. Then the final task was to combine them all together into one calculation. The lab requires extensive use of the RJMP and RET commands to implement all the different subroutines. Simulation was used to confirm that our process was correct and to view each of the operands in memory. In this lab we used the same process from lab 4 to move the values from program memory to data memory. Breakpoints were set to view each operand being loaded into data memory and when the results of the subtraction or addition were stored.

Program Overview

This program can really be broken down into its initialization and basic function calls, all of which rely on the use of the stack to be able to make several reference calls throughout the program. What this means is at the beginning of the program, we begin with the initialization of the stack pointer. From here, we will begin building our first component, operand loading from program memory into specified data memory. First, bytes were assigned to specific locations that we hoped to use with the specified operations. From here, the operands had to be written as data words in memory. This is so we had them written in program memory and prepared for further transfer. From here, per each operation, we would need to shift the values from program memory to the specified bytes we talked about before. This means writing the address of the storage location in one address register (Y), and the address in program memory to another address register (Z), loading from data memory into a multi purpose register, incrementing to prepare to grab the next one, rinsing repeating for every operand at any data point.

Now that we have established the methodology we will use to grab the values, we must discuss their usage. Relative call, or rcall operations will be used to call the functions that we had created later into the program portions. These portions contain a return call that will use the stack to return. Add ans subtract are based on using operations with carries to manipulate multiple bytes, mul24 is a modification on mu16, with more carry calls and loops for different addition use, and compound brings all of the ideas together into one large arithmetic problem

Init

This INIT routine is much shorter compared to previous labs. For lab 5 we simply initialized the stack pointer and placed it at the end of the SRAM. Additionally, the zero register was cleared because it was to be used in the arithmetic.

Add 16

Add 16 takes two 16 bit numbers, adds them, and generates a 24 bit result. The third byte of the result is the final carry from adding the second byte of the operands. It begins by pointing X, Y, and Z to the operands and the result. From there, we add the operands byte by byte and take care to use adc for the final two sequence of addition. If a carry is generated when the first byte is added it is used in the following calculation. The final carry out is held in the last byte of the result. We finish by storing the results to the address held by Z.

Sub 16

The process for Sub 16 is similar to add except the result in the case will not be 24 bit because we are guaranteed a 16 bit result; There cannot be a carry that results in a third byte for the result. It begins the same way as the

previous subroutine with X, Y, and Z pointing to the operands and the result. The subtraction is performed on each byte with the byte wise subtraction using the carry bit. Again, we store the result in the address pointed to by Z.

MUL 24

Mul 24 was the most complicated of the subroutines and generates a 48 bit result given two 24 bit operands. This operation is performed in a byte wise manner and requires 9 iterations to generate the proper result. Each byte of one operand must be multiplied by the three operands of the other, meaning 9 total iterations for 24 bit operands. It begins by pointing X, Y, and Z to the operands and results that have been reserved in memory. They will be incremented throughout the course of the calculation to perform the multiplication for each byte. X is moved back to the start of addrA at the beginning of each ILOOP and then each byte is multiplied by the individual bytes of Y. The inner loop begins with having one byte from each operand being loaded into A and B. They are multiplied and new values are given to A and B to repeat the process. Both bytes of the previous 2 results are added with a carry

COMPOUND

The compound function is really just a combination of function use by rcall and data memory manipulation. This is a large parallel in what we see in the main portion of the code, which largely composed of using the addressing registers to take data words declared in the data memory, and placing them from the position that they have been declared in the program memory, and then making use of them in the data memory positions that have been set aside specifically for the operations that we hope to conduct. We will start by initializing subtract operands and the first addition operands, specified by name D, E and F before making our initial compound call. Then, any data manipulation of the resultant values will be conducted in the compound function itself.

ADDITIONAL QUESTIONS

1. Although we dealt with unsigned numbers in this lab, the ATmega128 microcontroller also has some features which are important for performing signed arithmetic. What does the V flag in the status register indicate? Give an example (in binary) of two 8-bit values that will cause the V flag to be set when they are added together

   The V flag indicates an overflow was present in the last calculation. It is used exclusively with two's complement operations and does not indicate anything for unsigned math. The carry bit is the unsigned counterpart to the overflow bit and is not relevant to signed calculations. If using signed numbers, the calculation below would result in the overflow bit being turned on. The two operands are negative and thus the result should be negative but is not.

   1000 0000
   + 1000 0000
   _____
   0000 0000

2. . In the skeleton file for this lab, the .BYTE directive was used to allocate some data memory locations for MUL16's input operands and result. What are some benefits of using this directive to organize your data memory, rather than just declaring some address constants using the .EQU directive?

   EQU only allows us to attach a label to a constant or address. You cannot determine the size of memory you want allocated and are just limited to the size of the constant or address. If we want to create a

variable to be used .BYTE is the correct choice. .BYTE allows us to set aside a desired number of bytes for structures or variables. .BYTE is used for arrays and can attach a label to a set amount of reserved bytes. These bytes must be initialized by the programmer and can be referred to by their high and low bytes.

Difficulties

This lab was quite difficult and took much longer than previous labs. Initially we were having trouble beginning the simulation because we forgot to save before starting the debug process. Once, this was corrected we confirmed that our addition was correct and began adapting that subroutine for subtraction. Developing MUL 24 was quick but finding our why our final carry was missing was time consuming and frustrating. The hardest part of the lab getting the combination subroutine to work properly. We broke MUL 24 in the process which added to the frustration, but we finally found out that we need to clear the result of the previous test of MUL 24 before used the operands D, E, and F.

Conclusion

This lab was the most challenging so far, but the skills and concepts used in this lab are very important for success in this class. This lab required extensive memory management and knowledge of subroutines. Implementing the add and subtract functions were the most straightforward parts of this lab. Combining all of the subroutines together and managing all the operands and results from the operations gave us the most trouble. It was valuable to see how managing the carry bit can using 8 bit addition or subtraction can be scaled up to larger operations. As we progress through each of the labs we've begun to see how useful the simulation tools are and being able to see the state of the program at a glance. It has helped to develop our problem solving skills when using Atmel, and to better understand writing in assembly .

Source Code

```
;***********************************************************

;*

;*      Faaiq_Waqar_and_Jordan_Brown_Lab

;*      11/6/2019

;*      This program uses 3 subroutines to perform arithmetic

;*      There is 16 bit addition and subtraction as well as 24 bit multiplication

;*      The compound function uses each of these subroutines to perform ((opD-opE)-opF)*2

;*

;*

;*

;***********************************************************

;               Faaiq Waqar and Jordan Brown

;*              11/5/2019
```

```
;*
;*
;***********************************************************


.include "m128def.inc"                    ; Include definition file


;***********************************************************
;*      Internal Register Definitions and Constants
;***********************************************************
.def    mpr = r16                         ; Multipurpose register
.def    rlo = r0                          ; Low byte of MUL result
.def    rhi = r1                          ; High byte of MUL result
.def    zero = r2                         ; Zero register, set to zero in INIT, useful for
calculations
.def    A = r3                            ; A variable
.def    B = r4                            ; Another variable


.def    oloop = r17                       ; Outer Loop Counter
.def    iloop = r18                       ; Inner Loop Counter




;***********************************************************
;*      Start of Code Segment
;***********************************************************
.cseg                                             ; Beginning of code segment


;-----------------------------------------------------------
; Interrupt Vectors
;-----------------------------------------------------------
.org    $0000                             ; Beginning of IVs
```

```
                rjmp    INIT

.org    $0046                           ; End of Interrupt Vectors

;-----------------------------------------------------------
; Program Initialization
;-----------------------------------------------------------
INIT:                                           ; The initialization routine
                ; Initialize Stack Pointer
                ; TODO
                ldi r16, low(RAMEND)  ; load low bits of RAMEND into r16
                out SPL, r16                ; output r16 into stack pointer low
                ldi r16, high(RAMEND) ; load high bits of RAMEND into r16
                out SPH, r16                ; output r16 into stack pointer high
                ; Init the 2 stack pointer registers

                clr             zero                ; Set the zero register to zero, maintain
                                                    ; these semantics, meaning, don't
                                                    ; load anything else into it.

;-----------------------------------------------------------
; Main Program
;-----------------------------------------------------------
MAIN:                                           ; The Main program
                ; Setup the ADD16 function direct test
                                ldi             YL, low($0110)              ; Load Y with address
of operand in data mem
                                ldi             YH, high($0110)                     ; For low and
high bits
                                ldi             ZL, low(OperandA<<1)  ; Use operand A and place
from prog mem addr
```

```
                            ldi         ZH, high(OperandA<<1) ; Into Z, using the high and
low bit placements

                            lpm         r16, Z+                         ;   Load   the
first byte from program mem to r16, post-inc

                            st          Y+, r16                         ;  store  with
post increment into data memory at Y

                            lpm         r16, Z

                            st          Y+, r16                         ; Repeat  this
process for the second set of bytes for Op A

                            ldi         ZL, low(OperandB<<1)  ;  Use  operand  B  and  place
from prog mem addr

                            ldi         ZH, high(OperandB<<1) ; Into Z, using high and low
bit placements

                            lpm         r16, Z+                         ;   Load   the
first byte from program mem to r16, post-inc

                            st          Y+, r16                         ;  store  with
post increment into data memory at Y

                            lpm         r16, Z+

                            st          Y+, r16                         ;   Rinse   and
repeat for the second byte


                            ; Move values 0xFCBA and 0xFFFF in program memory to data memory

                            ; memory locations where ADD16 will get its inputs from

                            ; (see "Data Memory Allocation" section below)


                nop ; Check load ADD16 operands (Set Break point here #1)

                            ; Call ADD16 function to test its correctness

                            ; (calculate FCBA + FFFF)

                            rcall   ADD16


                nop ; Check ADD16 result (Set Break point here #2)

                            ; Observe result in Memory window


               ; Setup the SUB16 function direct test
```

```
                            ldi         YL, low($0130)              ;  Load  Y  with  the
address of operand in data
                            ldi         YH, high($0130)                     ;  For  low  and
high bits
                            ldi         ZL, low(OperandC<<1)  ;  Use  operand  C  and  place
from prog mem addr
                            ldi         ZH, high(OperandC<<1) ; into Z, using high and low
byte placements
                            lpm         r16, Z+                             ;  Load  Z  from
Program Memory, post inc
                            st          Y+, r16                             ;   store   the
byte into Y for data mem, post inc
                            lpm         r16, Z                              ;   Rinse   and
repeat
                            st          Y+, r16                             ;    Accounting
for the second byte of the operand
                            ldi         ZL, low(OperandG<<1)  ; Load  Z  with  OperandG,  the
second operand for
                            ldi         ZH, high(OperandG<<1) ;   the   subtract   16   bit
funnction
                            lpm         r16, Z+                             ;    Load    the
data byte into r16, post inc
                            st          Y+, r16                             ;  Store  into
data memory at Y, post inc
                            lpm         r16, Z+                             ;   Rinse   and
repear for the second
                            st          Y+, r16                             ; storage byte
into data memory
                            ; Move values 0xFCB9 and 0xE420 in program memory to data memory
                            ; memory locations where SUB16 will get its inputs from


                nop ; Check load SUB16 operands (Set Break point here #3)
                            ; Call SUB16 function to test its correctness
                            ; (calculate FCB9 - E420)
                            rcall   SUB16
```

```
                nop ; Check SUB16 result (Set Break point here #4)
                            ; Observe result in Memory window


              ; Setup the MUL24 function direct test
                            ldi         YL, low($0100)              ;  Take  the  operand
Value in data mem
                            ldi         YH, high($0100)                    ;   and   place
into YH:YL regs for storage
                            ldi         ZL, low(OperandX<<1)  ; Take the operand value in
program mem
                            ldi         ZH, high(OperandX<<1) ; and place into the ZH:ZL
regs for movement
                            lpm         r16, Z+                            ;   load   from
program memory, take Z operand
                            st          Y+, r16                            ;   store   the
value in r16 from pm to Y dm
                            lpm         r16, Z+                            ;   load   from
program memory, take Z operand
                            st          Y+, r16                            ;   store   the
value from r16 from pm to Y dm
                            lpm         r16, Z+                            ;   Rinse   and
repead
                            st          Y+, r16                            ; Store Final
Value


                            ldi         YL, low($0103)              ;  Take  the  operand
Value in data mem
                            ldi         YH, high($0103)                    ;   and   place
into YH:YL regs for storage
                            ldi         ZL, low(OperandX<<1)  ; Take the operand value in
program mem
                            ldi         ZH, high(OperandX<<1) ; and place into the ZH:ZL
regs for movement
                            lpm         r16, Z+                            ;   load   from
program memory, take Z operand
                            st          Y+, r16                            ;   store   the
value in r16 from pm to Y dm
```

```
                            lpm             r16, Z+                             ;   load   from
program memory, take Z operand
                            st              Y+, r16                             ;   store   the
value from r16 from pm to Y dm
                            lpm             r16, Z+                             ;   Rinse   and
repead
                            st              Y+, r16                             ;  Store  Final
Value


                            ; Move values 0xFFFFFF and 0xFFFFFF in program memory to data
memory
                            ; memory locations where MUL24 will get its inputs from


                nop ; Check load MUL24 operands (Set Break point here #5)
                            ; Call MUL24 function to test its correctness
                            ; (calculate FFFFFF * FFFFFF)
                            rcall MUL24


                nop ; Check MUL24 result (Set Break point here #6)
                            ; Observe result in Memory window


               ; Call the COMPOUND function
                            ldi             YL, low($0130)                ; Set up The address
                            ldi             YH, high($0130)                     ;  And  prepare
into the Y registers
                            ldi             ZL, low(OperandD<<1)  ;  Prepare  pm  for  storage  in
compoun
                            ldi             ZH, high(OperandD<<1) ;
                            lpm             r16, Z+                             ;  load  z  from
program mem into r16
                            st              Y+, r16                             ;   store   r16
into Y, repeat this for
                            lpm             r16, Z                              ;  next  set  of
byte
```

```
                        st          Y+, r16                         ;
                        ldi         ZL, low(OperandE<<1)  ; Repat the process for the
next process
                        ldi         ZH, high(OperandE<<1) ; Storing Operand E into Data
Memory
                        lpm         r16, Z+                         ;
                        st          Y+, r16                         ;
                        lpm         r16, Z+                         ;
                        st          Y+, r16                         ;


                        ldi         YL, low($0110)              ;  Prepare  Operand  F
for addition
                        ldi         YH, high($0110)                 ; Operation
                        ldi         ZL, low(OperandF<<1)  ;
                        ldi         ZH, high(OperandF<<1) ;
                        lpm         r16, Z+                         ;
                        st          Y+, r16                         ;
                        lpm         r16, Z+                         ;
                        st          Y+, r16                         ;


                        nop ; Check load COMPOUND operands (Set Break point here #7)


                        rcall COMPOUND


              nop ; Check COMPUND result (Set Break point here #8)
                        ; Observe final result in Memory window


DONE:   rjmp    DONE                ; Create an infinite while loop to signify the
                                                    ; end of the program.


;***********************************************************
```

```
;*      Functions and Subroutines
;*************************************************************


;-------------------------------------------------------------
; Func: ADD16
; Desc: Adds two 16-bit numbers and generates a 24-bit number
;              where the high byte of the result contains the carry
;              out bit.
;-------------------------------------------------------------
ADD16:
               ; Load beginning address of first operand into X
               ldi            XL, low(ADD16_OP1)    ; Load low byte of address
               ldi            XH, high(ADD16_OP1)   ; Load high byte of address

               ; Load beginning address of second operand into Y
               ldi            YL, low(ADD16_OP2)    ; Load low byte of address
               ldi            YH, high(ADD16_OP2)   ; Load high byte of address
               ; Load beginning address of result into Z
               ldi            ZL, low(ADD16_Result) ; Load low byte of address
               ldi            ZH, high(ADD16_Result)       ; Load high byte of address

               ; Execute the function
               ld             r16, X+                      ;  Load  r16  with  first  byte  of
Operand 1, post inc
               ld             r17, Y+                      ;  Load  r17  with  first  byte  of
Operand 2, post inc
               add            r17, r16                     ; add the contents of r16 and r17
together
               st             Z+, r17                      ; store the first resultant into Z,
post inc
               ld             r16, X                       ; load r16 with the second byte of
Operand 1
```

```
                ld              r17, Y                          ; load r17 with the second byte of
Operand 2
                adc             r17, r16                        ;  add  with  carry  from  previous
operation of r16/17
                st              Z+, r17                         ; store the second resultant into Z,
post inc
                ldi             r16, $00                        ; store 0 into r16
                ldi             r17, $00                        ; store 0 into r17
                adc             r17, r16                        ;  add  to  see  if  there  is  a  single
carry bit left
                st              Z, r17                          ;  store  the  result  into  the  Z
pointer data mem


                ret                                             ; End a function with RET


;-----------------------------------------------------------
; Func: SUB16
; Desc: Subtracts two 16-bit numbers and generates a 16-bit
;               result.
;-----------------------------------------------------------
SUB16:
                ; Execute the function here
                ldi             XL, low(SUB16_OP1)    ; Load low byte of address
                ldi             XH, high(SUB16_OP1)   ; Load high byte of address


                ; Load beginning address of second operand into Y
                ldi             YL, low(SUB16_OP2)
                ldi             YH, high(SUB16_OP2)
                ; Load beginning address of result into Z
                ldi             ZL, low(SUB16_Result)
                ldi             ZH, high(SUB16_Result)
```

```
                ; Execute the function
                ld              r16, X+                 ; Take X operand, place val in r16, post
inc
                ld              r17, Y+                 ; Take Y operand, place val in r17, post
inc
                sub             r16, r17                ; Subtract the value, first from second one
                st              Z+, r16                 ; Store the resultand into Z, post inc
                ld              r16, X                  ; Take X operand, place val into r16,
second byte
                ld              r17, Y                  ; Take Y operand, place val into r17,
second byte
                sbc             r16, r17                ; Subtract with carry for the two ops
                st              Z, r16                  ; Store the resutant into Z


                ret                                             ; End a function with RET


;-----------------------------------------------------------
; Func: MUL24
; Desc: Multiplies two 24-bit numbers and generates a 48-bit
;               result.
;-----------------------------------------------------------
MUL24:
                ; Execute the function here
                push    A                               ; Save A register
                push    B                               ; Save B register
                push    rhi                             ; Save rhi register
                push    rlo                             ; Save rlo register
                push    zero                    ; Save zero register
                push    XH                              ; Save X-ptr
                push    XL
                push    YH                              ; Save Y-ptr
                push    YL
```

```
                push    ZH                          ; Save Z-ptr
                push    ZL
                push    oloop               ; Save counters
                push    iloop

                clr             zero                ; Maintain zero semantics

                ; Set Y to beginning address of B
                ldi             YL, low(addrB) ; Load low byte
                ldi             YH, high(addrB)       ; Load high byte

                ; Set Z to begginning address of resulting Product
                ldi             ZL, low(LAddrP)       ; Load low byte
                ldi             ZH, high(LAddrP); Load high byte

                ; Begin outer for loop
                ldi             oloop, 3              ; Load counter
MUL24_OLOOP:
                ; Set X to beginning address of A
                ldi             XL, low(addrA) ; Load low byte
                ldi             XH, high(addrA)       ; Load high byte

                ; Begin inner for loop
                ldi             iloop, 3              ; Load counter
MUL24_ILOOP:
                ld              A, X+               ; Get byte of A operand
                ld              B, Y                ; Get byte of B operand
                mul             A,B                         ; Multiply A and B
                ld              A, Z+               ; Get a result byte from memory
                ld              B, Z+               ; Get the next result byte from memory
```

```
add             rlo, A                  ; rlo <= rlo + A
adc             rhi, B                  ; rhi <= rhi + B + carry
ld              A, Z                    ; Get a third byte from the result
adc             A, zero                 ; Add carry to A
st              Z+, A                   ; Store third byte to memory
ld              A, Z                    ; Get a third byte from the result
adc             A, zero                 ; Add carry to A
st              Z, A                    ; Store third byte to memory
sbiw    ZH:ZL, 1                ; Z <= Z - 2
st              -Z, rhi                 ; Store second byte to memory
st              -Z, rlo                 ; Store first byte to memory
adiw    ZH:ZL, 1                ; Z <= Z + 1
dec             iloop                   ; Decrement counter
brne    MUL24_ILOOP             ; Loop if iLoop != 0
; End inner for loop

sbiw    ZH:ZL, 2                ; Z <= Z - 2
adiw    YH:YL, 1                ; Y <= Y + 1
dec             oloop                   ; Decrement counter
brne    MUL24_OLOOP             ; Loop if oLoop != 0

; End outer for loop

pop             iloop                   ; Restore all registers in reverves order
pop             oloop
pop             ZL
pop             ZH
pop             YL
pop             YH
pop             XL
```

```
                pop             XH
                pop             zero
                pop             rlo
                pop             rhi
                pop             B
                pop             A

                ret                                     ; End a function with RET

;-----------------------------------------------------------
; Func: COMPOUND
; Desc: Computes the compound expression ((D - E) + F)^2
;               by making use of SUB16, ADD16, and MUL24.
;
;               D, E, and F are declared in program memory, and must
;               be moved into data memory for use as input operands.
;
;               All result bytes should be cleared before beginning.
;-----------------------------------------------------------
COMPOUND:

                ; Setup SUB16 with operands D and E
                ; Perform subtraction to calculate D - E
                rcall SUB16
                ; Setup the ADD16 function with SUB16 result and operand F
                ; Perform addition next to calculate (D - E) + F
                ldi             YL, low($0112)        ;
                ldi             YH, high($0112)             ;
                ldi             ZL, low($0140)        ;
                ldi             ZH, high($0140)             ;
```

```
ld              r16, Z+                         ;
st              Y+, r16                         ;
ld              r16, Z+                         ;
st              Y+, r16                         ;


rcall ADD16
; Setup the MUL24 function with ADD16 result as both operands
; Perform multiplication to calculate ((D - E) + F)^2
ldi             YL, low($0100)          ;
ldi             YH, high($0100)             ;
ldi             ZL, low($0120)          ;
ldi             ZH, high($0120)             ;
ld              r16, Z+                         ;
st              Y+, r16                         ;
ld              r16, Z+                         ;
st              Y+, r16                         ;
ld              r16, Z+                         ;
st              Y+, r16                         ;


ldi             YL, low($0103)          ;
ldi             YH, high($0103)             ;
ldi             ZL, low($0120)          ;
ldi             ZH, high($0120)             ;
ld              r16, Z+                         ;
st              Y+, r16                         ;
ld              r16, Z+                         ;
st              Y+, r16                         ;
ld              r16, Z+                         ;
st              Y+, r16                         ;
```

```
                ldi             ZL, low(LAddrP)             ; Clear out previous multiplication
value
                ldi             ZH, high(LAddrP)      ; By Storing the reg with value
                ldi             r16, $00                    ;  00,  then,  changing  each  of  the
result values
                st              Z+, r16                     ; with input as the given value
                st              Z+, r16                     ; again
                st              Z+, r16                     ; again
                st              Z+, r16                     ; again
                st              Z+, r16                     ; again
                st              Z+, r16                     ; again


                rcall MUL24


                ret                                         ; End a function with RET


;-----------------------------------------------------------
; Func: MUL16
; Desc: An example function that multiplies two 16-bit numbers
;                       A - Operand A is gathered from address $0101:$0100
;                       B - Operand B is gathered from address $0103:$0102
;                       Res - Result is stored in address
;                                       $0107:$0106:$0105:$0104
;               You will need to make sure that Res is cleared before
;               calling this function.
;-----------------------------------------------------------
MUL16:
                push    A                           ; Save A register
                push    B                           ; Save B register
                push    rhi                         ; Save rhi register
                push    rlo                         ; Save rlo register
```

```
                push    zero                    ; Save zero register
                push    XH                              ; Save X-ptr
                push    XL
                push    YH                              ; Save Y-ptr
                push    YL
                push    ZH                              ; Save Z-ptr
                push    ZL
                push    oloop                   ; Save counters
                push    iloop

                clr             zero                    ; Maintain zero semantics

                ; Set Y to beginning address of B
                ldi             YL, low(addrB) ; Load low byte
                ldi             YH, high(addrB)         ; Load high byte

                ; Set Z to begginning address of resulting Product
                ldi             ZL, low(LAddrP)         ; Load low byte
                ldi             ZH, high(LAddrP); Load high byte

                ; Begin outer for loop
                ldi             oloop, 2                ; Load counter
MUL16_OLOOP:
                ; Set X to beginning address of A
                ldi             XL, low(addrA) ; Load low byte
                ldi             XH, high(addrA)         ; Load high byte

                ; Begin inner for loop
                ldi             iloop, 2                ; Load counter
MUL16_ILOOP:
```

```
ld            A, X+                ; Get byte of A operand
ld            B, Y                 ; Get byte of B operand
mul           A,B                        ; Multiply A and B
ld            A, Z+                ; Get a result byte from memory
ld            B, Z+                ; Get the next result byte from memory
add           rlo, A               ; rlo <= rlo + A
adc           rhi, B               ; rhi <= rhi + B + carry
ld            A, Z+                ; Get a third byte from the result

adc           A, zero              ; Add carry to A
st            Z, A                 ; Store third byte to memory
st            -Z, rhi              ; Store second byte to memory
st            -Z, rlo              ; Store first byte to memory
adiw   ZH:ZL, 1             ; Z <= Z + 1
dec           iloop                ; Decrement counter
brne   MUL16_ILOOP          ; Loop if iLoop != 0
; End inner for loop

sbiw   ZH:ZL, 1             ; Z <= Z - 1
adiw   YH:YL, 1             ; Y <= Y + 1
dec           oloop                ; Decrement counter
brne   MUL16_OLOOP          ; Loop if oLoop != 0
; End outer for loop

pop           iloop                ; Restore all registers in reverves order
pop           oloop
pop           ZL
pop           ZH
pop           YL
pop           YH
```

```
                pop             XL
                pop             XH
                pop             zero
                pop             rlo
                pop             rhi
                pop             B
                pop             A
                ret                                     ; End a function with RET


;-------------------------------------------------------------
; Func: Template function header
; Desc: Cut and paste this and fill in the info at the
;               beginning of your functions
;-------------------------------------------------------------
FUNC:                                           ; Begin a function with a label
                ; Save variable by pushing them to the stack

                ; Execute the function here

                ; Restore variable by popping them from the stack in reverse order
                ret                                     ; End a function with RET



;*************************************************************
;*      Stored Program Data
;*************************************************************

; Enter any stored data you might need here

; ADD16 operands
```

```
OperandA:
        .DW 0xFCBA                          ; Addition Operand A
OperandB:
        .DW 0XFFFF                          ; Addition Operand B
; SUB16 operands
OperandC:
        .DW 0XFCB9                          ; Subtraction Operand C
OperandG:
        .DW 0XE420                          ; Subtraction Operand G
; MUL24 operands
OperandX:
        .DW 0XFFFFFF                ; Multiplication Operand X
OperandY:
        .DW 0XFFFFFF                ; Multiplication Operand Y
; Compoud operands
OperandD:
        .DW    0xFCBA                       ; test value for operand D
OperandE:
        .DW    0x2019                       ; test value for operand E
OperandF:
        .DW    0x21BB                       ; test value for operand F


;***********************************************************
;*      Data Memory Allocation
;***********************************************************


.dseg
.org   $0100                        ; data memory allocation for MUL16 example
addrA: .byte 3
addrB: .byte 3                      ; Changed to 3 bytes for addr and 6 bytes for LAddrP
```

```
LAddrP: .byte 6


; Below is an example of data memory allocation for ADD16.
; Consider using something similar for SUB16 and MUL24.


.org   $0110                        ; data memory allocation for operands
ADD16_OP1:
               .byte 2                      ; allocate two bytes for first operand of ADD16
ADD16_OP2:
               .byte 2                      ; allocate two bytes for second operand of ADD16


.org   $0120                        ; data memory allocation for results
ADD16_Result:
               .byte 3                      ; allocate three bytes for ADD16 result


.org   $0130                        ; set origin point for subtraction operations
SUB16_OP1:
               .byte 2                      ; allocate two byes for first subtract operand
SUB16_OP2:
               .byte 2                      ; allocate two byes for second subtract operand
.org   $0140
SUB16_Result:                       ; allocate three bytes for the result of the funcion
               .byte 3
;***********************************************************
;*     Additional Program Includes
;***********************************************************
; There are no additional file includes for this program
```